

# Rapport

## optimisation continue

*Problème de Rosenbrock*

---

**WANG ZUNZUN**

**LYU YISHUO**

Cahier des charges :

Le but : Trouver la solution au problème de Rosenbrock.

L'outil utilisé : Matlab.

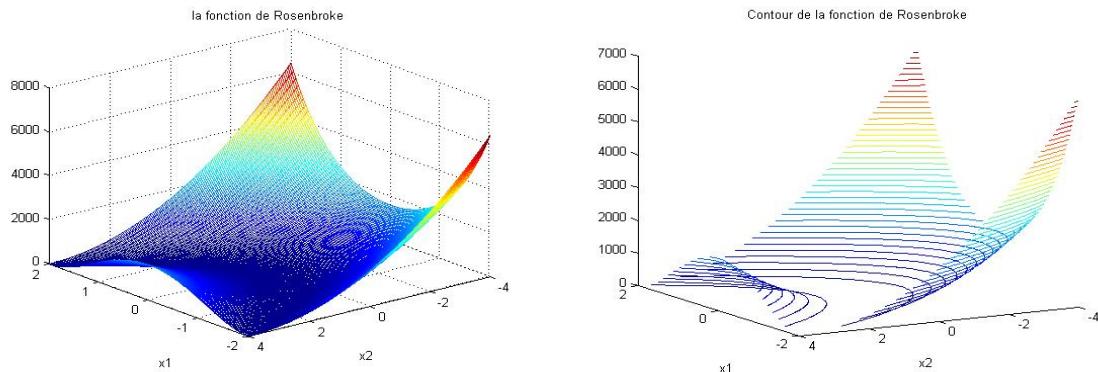
Les algorithmes : La méthode des plus fortes pentes avec préconditionnement,

La méthode de Newton avec recherche linéaire,

La méthode quasi-Newton BFGS.

## Question1 :

Représenter la fonction de Rosenbrock  $f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$  pour  $-2 < x_1 < 2, -4 < x_2 < 4$ .



On peut voir que le minimum local de cette fonction se situe approximativement au point (1,1) grâce aux images ci-dessus. Par la suite, nous allons utiliser les trois méthodes pour trouver le minimum local et nous allons comparer l'efficacité de ces trois méthodes.

\*Les codes pour représenter les deux images ci-dessus sont dans les fichiers **Test\_LaPlusForte.m**, **Test\_Newton\_Recherche\_Lineaire.m**, et **Test\_QuasiNewton.m**.

## Question2 :

### *La méthode des plus fortes pentes avec préconditionnement*

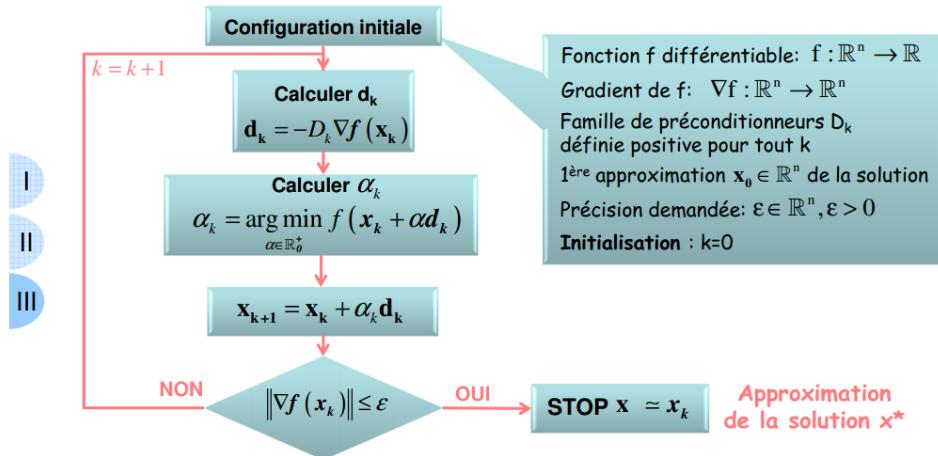
On utilise les fichiers suivants pour résoudre le problème :

**LaPlusForte.m, Choisir.m, CW1.m, CW2.m, Fonction.m**

### LaPlusForte.m : function x=LaPlusForte(x,beta1,beta2)

On a mis la fonction de Rosenbrock ( $f(x_1, x_2)$ ), le point de départ ( $x$ ), beta1 et beta2 en paramètre, on peut utiliser cette méthode pour obtenir les résultats affichés plus loin.

Cette fonction est basée sur l'algorithme vu en cours : (poly chapitres3 P15)



CW1.m, CW2. m :

```
function beta1=CW1(x,f,fgrad,alpha,d)
```

```
function beta2=CW2(x,fgrad,alpha,d)
```

Les deux méthodes ci-dessus décrivent respectivement la première condition de Wolfe et la deuxième condition de Wolfe. A chaque itération, on doit vérifier que les conditions de Wolfe sont respectées.

Fonction.m :

```
function v=Fonction(x,f)
```

On peut utiliser cette fonction pour calculer la valeur de  $f$ .

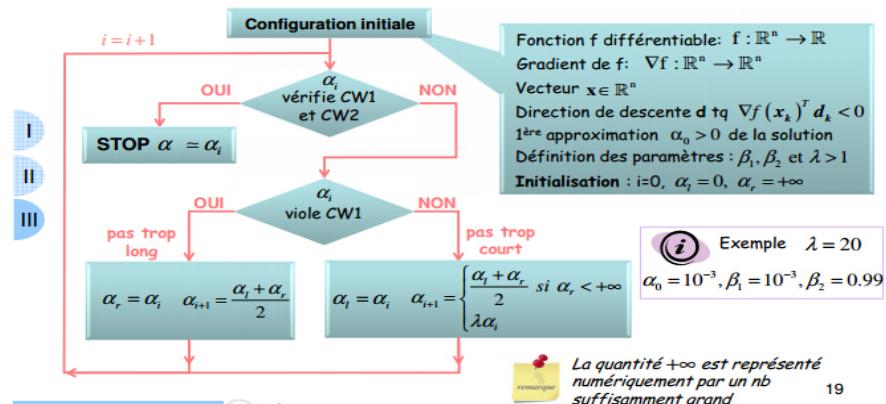
\*subs (f, variables, valeurs) renvoie une copie de la fonction  $f$  dans laquelle la fonction subs a remplacé toutes les occurrences des variables avec les valeurs spécifiées. Ensuite on retourne le résultat de l'opération.

Choisir.m :

```
function alpha=Choisir(x,f,fgrad,beta1,beta2,alpha,d)
```

Cette méthode réalise une recherche linéaire du pas vérifiant les conditions de Wolfe. Grâce à cela, on peut trouver un alpha optimisé dans chaque itération.

Cette méthode est basée sur l'algorithme de Fletcher-Lemaréchal cours: (poly chapitres3 P19)



\*La valeur par défaut : alpha initial égale 0.01.

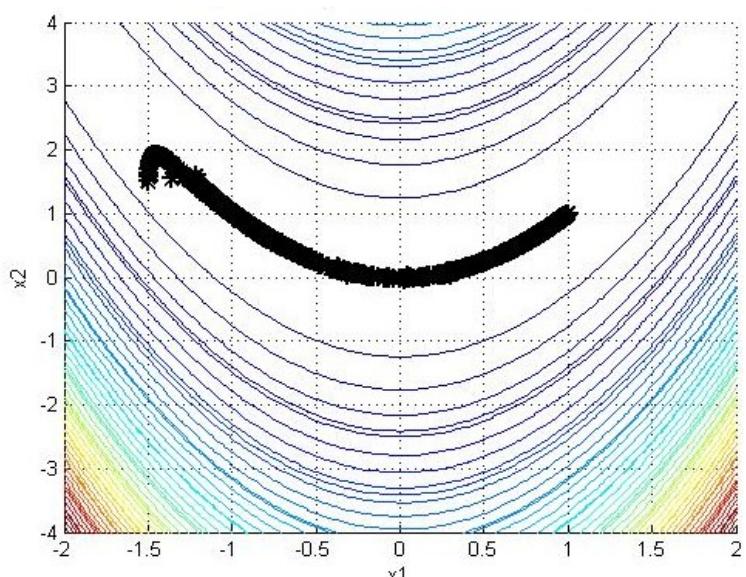
### CONDITION 1 :

beta1=0.1 ,beta2=0.4

Résultat: voir image ci-dessous

n=2181 itérations

Name	Value	Min	Max
X1	<161x81 double>	-2	2
X2	<161x81 double>	-4	4
f	<161x81 double>	0	6409
x	[ -1.5000:1.5000 ]	-1.50...	1.5000
x1	<1x81 double>	-2	2
x2	<1x161 double>	-4	4
y	[ 0.9900:0.9801 ]	0.9801	0.9900



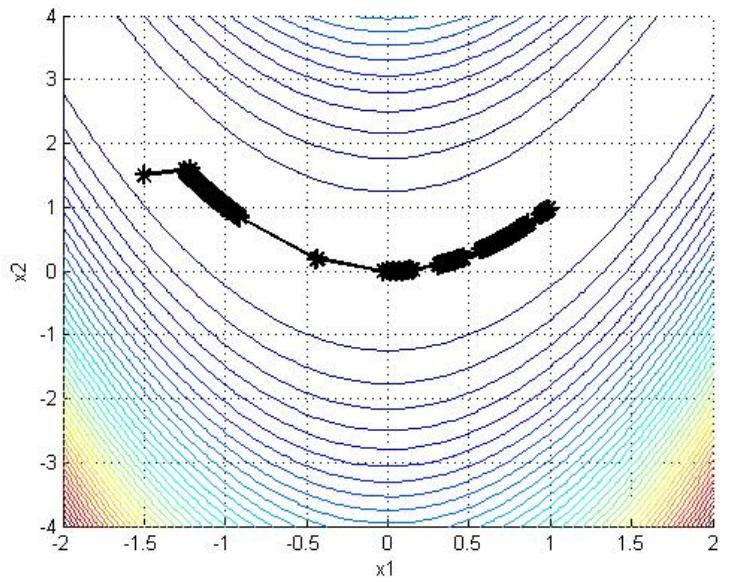
### CONDITION 2 :

beta1=0.3 beta2=0.7

Résultat: voir image ci-dessous

n=1603 itérations

Name	Value	Min	Max
X1	<161x81 double>	-2	2
X2	<161x81 double>	-4	4
f	<161x81 double>	0	6409
x	[ -1.5000; 1.5000 ]	-1.50...	1.5000
x1	<1x81 double>	-2	2
x2	<1x161 double>	-4	4
y	[ 0.9896; 0.9793 ]	0.9793	0.9896



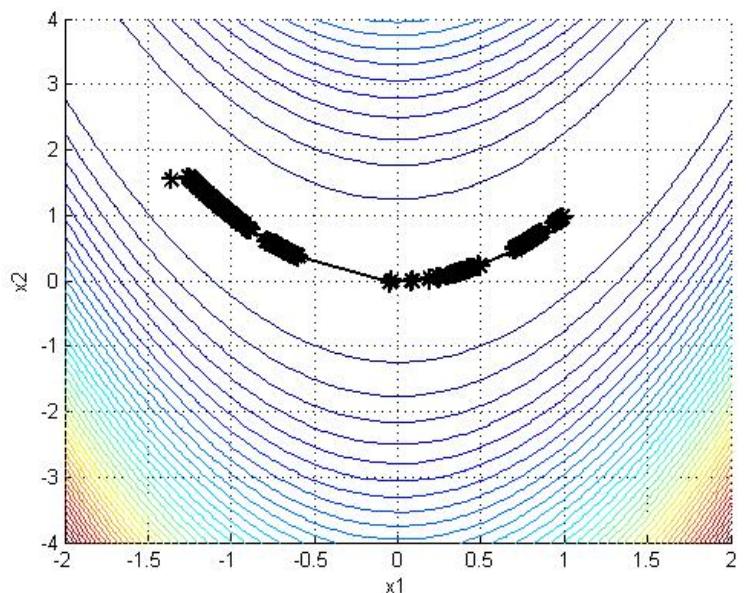
### CONDITION 3 :

beta1=0.4 beta2=0.7

Résultat: voir image ci-dessous

n=1479 itérations

Name	Value	Min	Max
X1	<161x81 double>	-2	2
X2	<161x81 double>	-4	4
f	<161x81 double>	0	6409
x	[ -1.5000; 1.5000 ]	-1.50...	1.5000
x1	<1x81 double>	-2	2
x2	<1x161 double>	-4	4
y	[ 0.9893; 0.9787 ]	0.9787	0.9893



## \*Etude de l'influence du choix point de départ

### CONDITION 1 :

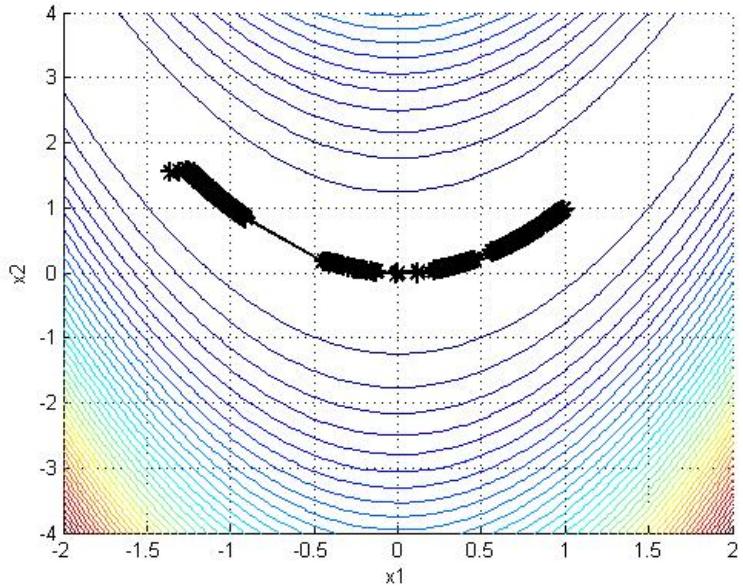
Pour  $x=[-1.5; 1.5]$ ;

$\beta_1=0.5, \beta_2=0.9$

Résultat: voir image ci-dessous

$n=1610$  itérations

Name	Value	Min	Max
X1	<161x81 double>	-2	2
X2	<161x81 double>	-4	4
f	<161x81 double>	0	6409
x	[ -1.5000; 1.5000 ]	-1.50...	1.5000
x1	<1x81 double>	-2	2
x2	<1x161 double>	-4	4
y	[ 0.9892; 0.9786 ]	0.9786	0.9892



### CONDITION 2 :

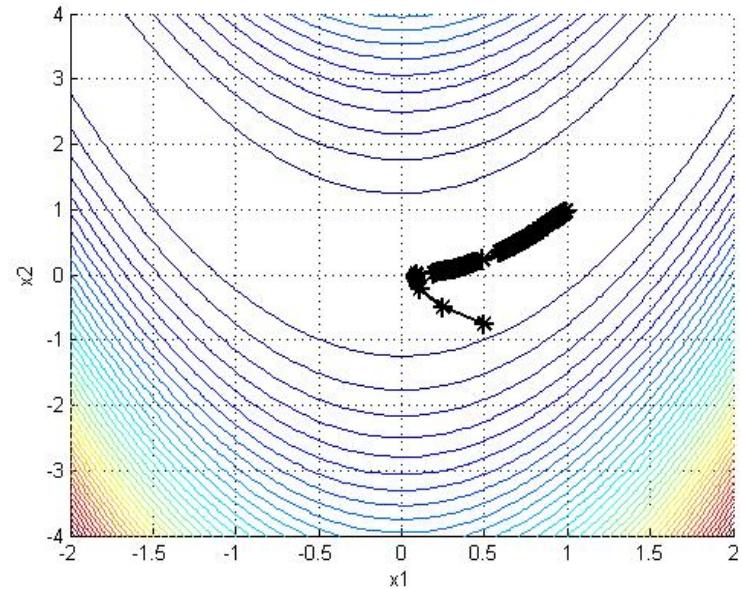
Pour  $x=[1; -1]$ ;

$\beta_1=0.5, \beta_2=0.9$

Résultat: voir image ci-dessous

$n=1437$  itérations

Name	Value	Min	Max
X1	<161x81 double>	-2	2
X2	<161x81 double>	-4	4
f	<161x81 double>	0	6409
x	[ 1;-1 ]	-1	1
x1	<1x81 double>	-2	2
x2	<1x161 double>	-4	4
y	[ 0.9892; 0.9785 ]	0.9785	0.9892



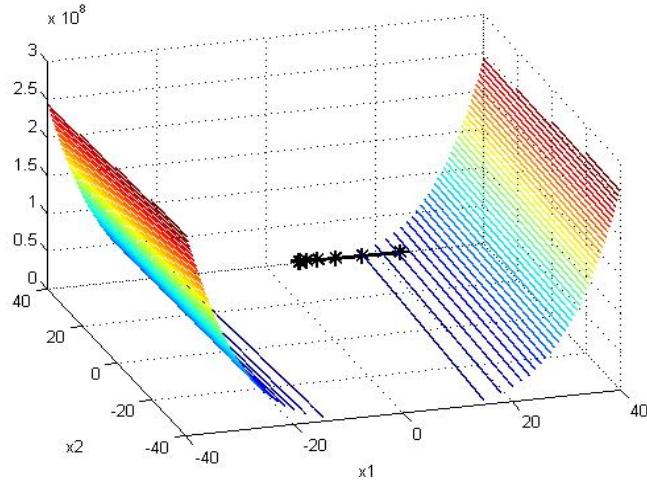
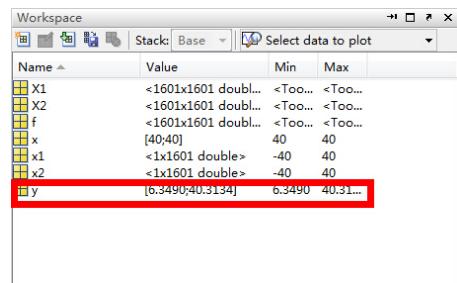
### CONDITION 3 :

Pour  $x=[40; 40]$ ;

$\beta_1=0.5$ ,  $\beta_2=0.9$

Résultat: voir image ci-dessous

$n=5000$  itérations



### Conclusion

Après exécution du programme, on remarque que l'on obtient bien un minimum global proche de  $(1,1)$ . Les différentes valeurs de  $\beta_1$ ,  $\beta_2$  et les différents points de départ influent sur le nombre d'itérations à réaliser pour arriver au résultat. Si le point de départ est trop éloigné du minimum, on ne converge pas.

Et quand

$\beta_1=0.1$ ,  $\beta_2=0.4$  il faut 2181 itérations pour arriver au point min  $(1,1)$  avec une marge d'erreur de 0.01.

$\beta_1=0.3$ ,  $\beta_2=0.7$  il faut 1603 itérations pour arriver au point min  $(1,1)$  avec une marge d'erreur de 0.01.

$\beta_1=0.4$ ,  $\beta_2=0.7$  il faut 1479 itérations pour arriver au point min  $(1,1)$  avec une marge d'erreur de 0.01.

D'après les résultats ci-dessus, si beta1 est trop proche de 0 ou si beta1 est trop proche de 1, il faut plus d'itération pour arriver au point min (1,1) avec une marge d'erreur de 0.01. En effet, le pas choisi n'est alors pas optimal.

Au contraire, quand beta1 et beta2 ne sont pas trop proches de 0 ou 1, c'est à dire que (beta2-beta1) est plus petit, la méthode est plus efficace. Il faut alors moins d'itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

Ensuite, on a fixé beta1=0.5, beta2=0.9.

Pour un point de départ :  $x=[-1.5; 1.5]$ , il faut 1610 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

Pour un point de départ :  $x=[1; -1]$ , il faut 1437 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

Pour un point de départ :  $x=[40; 40]$ , il faut plus de 5000 itérations, c'est-à-dire il ne converge pas vers le point min.

Donc quand le point de départ est plus proche du minimum, il faut moins d'itérations pour arriver au point min (1, 1) avec marge d'erreur de 0.01.

## Question3

### *La méthode de Newton avec recherche linéaire*

On calcule la matrice Hessienne de  $f$  avec le fichier Hessienne.m et la factorisation de Cholesky ( $L$ ) avec le fichier Factorisation.m et puis on utilise le résultat de cette factorisation pour calculer  $d$  à chaque itération de la boucle.

#### **Hessienne.m : $H=\text{Hessienne}(f)$**

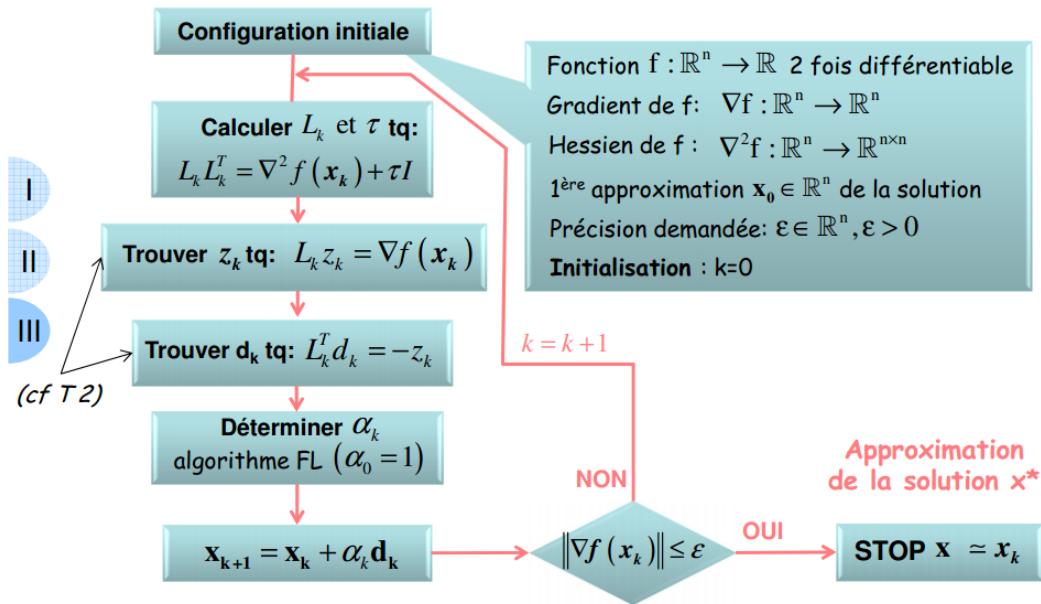
On peut calculer la matrice Hessienne de  $f$ .

#### **Factorisation.m : $L=\text{Factorisation}(h)$**

On réalise la factorisation de Cholesky avec la méthode chol() du logiciel Matlab.

#### **Newton.m : $x=\text{Newton}(x, \text{beta1}, \text{beta2})$**

Dans cette méthode on utilise l'algorithme suivant (poly chapitre3 P19) :

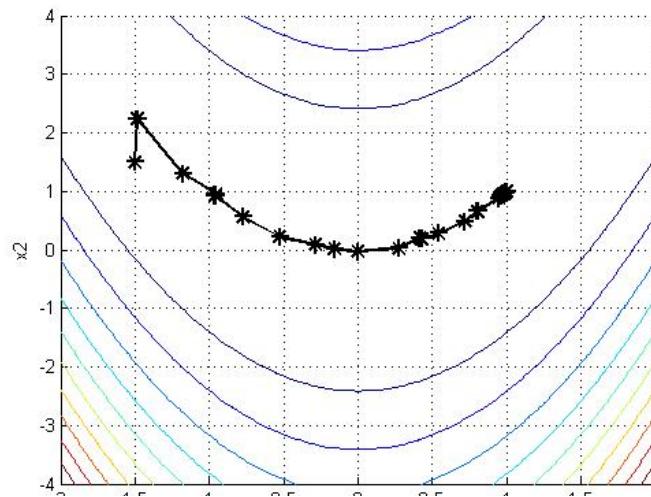
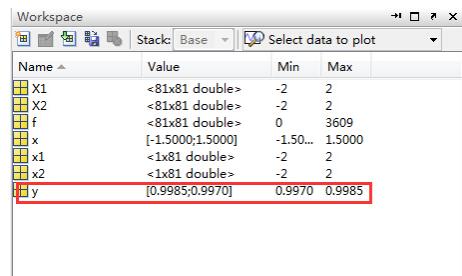


### CONDITION 1 :

Pour X=[-1.5;1.5];

beta1=0.1 beta2=0.9 ;

n=91.

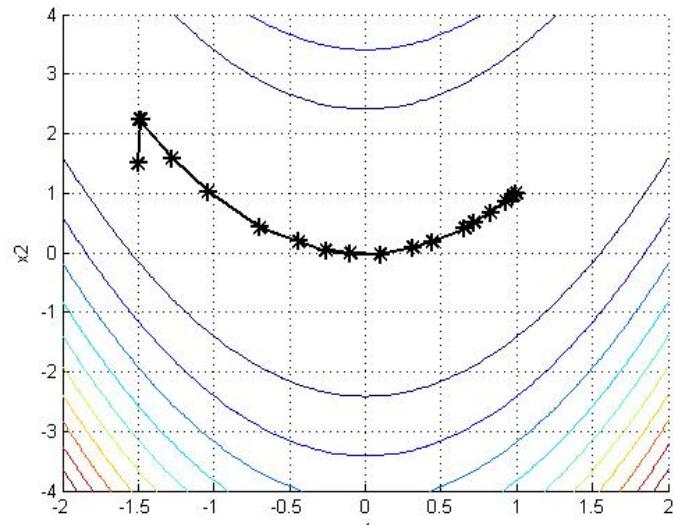
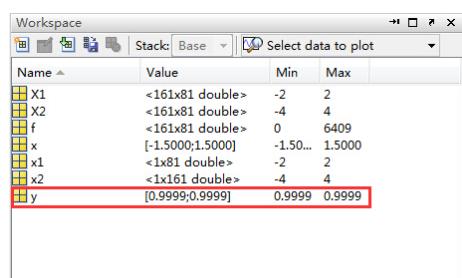


### CONDITION 2 :

Pour X=[-1.5;1.5];

beta1=0.4 beta2=0.7;

n=18.

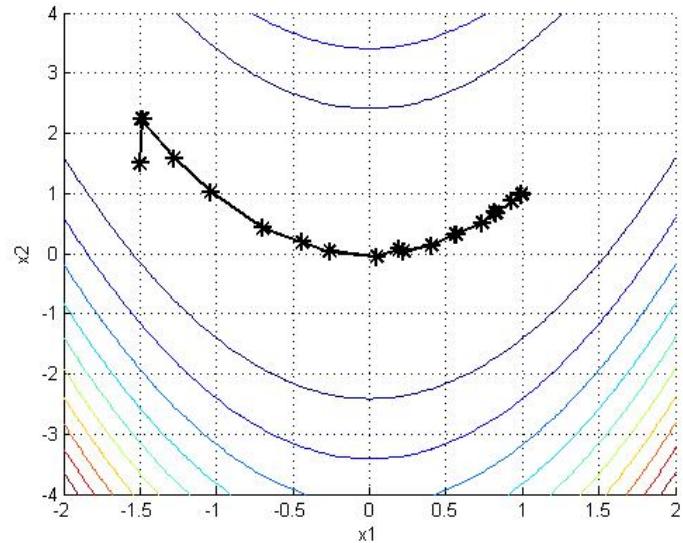
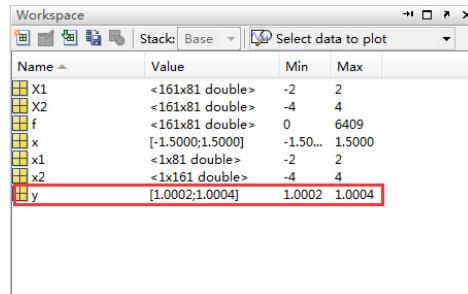


### CONDITION 3 :

Pour  $X=[-1.5;1.5]$ ;

$\beta_1=0.1$   $\beta_2=0.5$ ;

$n=20$ .

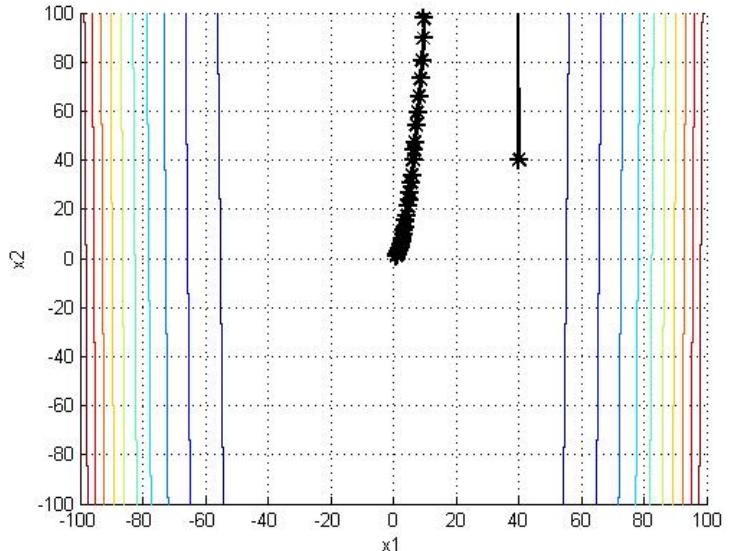
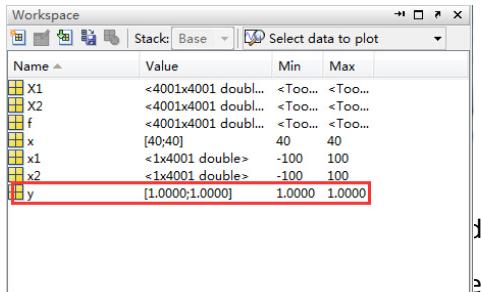


### CONDITION 4 :

Pour  $X=[40,40]$  ;

$\beta_1=0.1$   $\beta_2=0.5$ ;

$n=113$ .



## Conclusion

Le méthode de Newton avec recherche linéaire est plus efficace que le méthode des plus fortes pentes avec préconditionnement, parce qu'il faut moins d'itérations pour arriver au plus petit point (1,1).

Voici résultats que nous obtenons :

$\beta_1=0.1$ ,  $\beta_2=0.9$  il faut 91 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

beta1=0.1, beta2=0.5 il faut 20 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

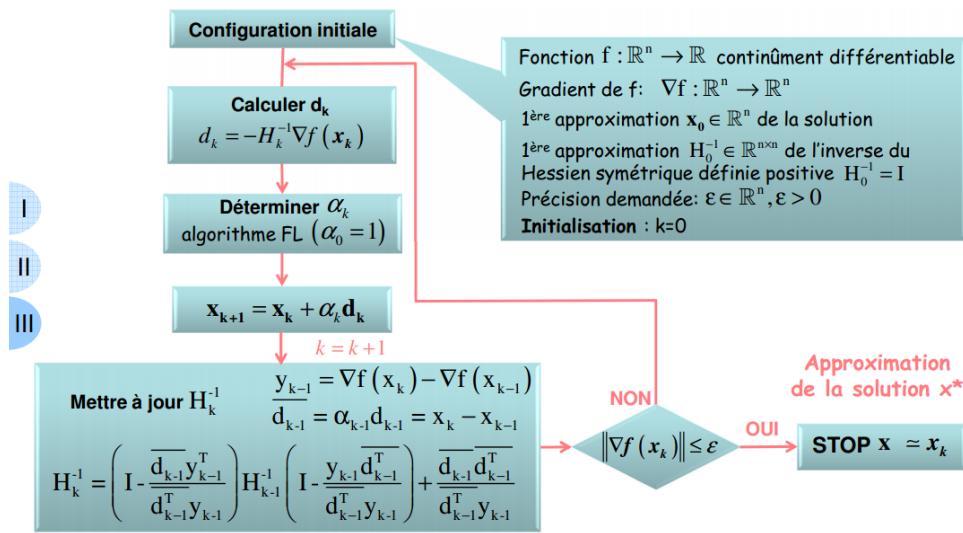
beta1=0.4, beta2=0.7 il faut 18 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

Quand ( $\beta_2 - \beta_1$ ) est plus grande, l'efficacité de la méthode de Newton avec recherche linéaire diminue (mais elle reste plus efficace que la méthode des plus fortes pentes avec préconditionnement). Quand ( $\beta_2 - \beta_1$ ) est petite, la méthode de Newton avec recherche linéaire est très efficace.

## Question4

### *La méthode de quasi-Newton BFGS*

1. Selon l'algorithme du cours (chapitre 3 T29)



29

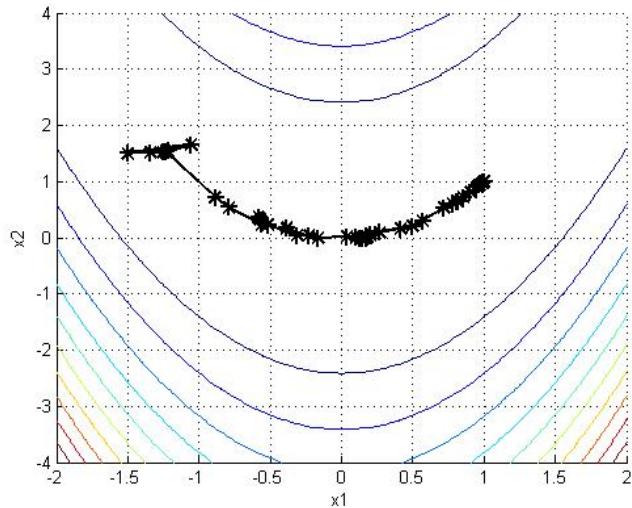
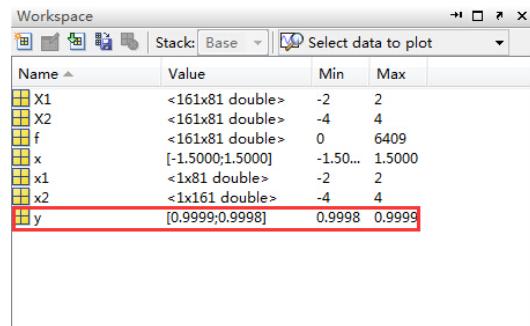
On met à jour  $H_k^{-1}$  à chaque itération (voir ci-dessus) après avoir mis à jour  $H_k$  avec la méthode de BFGS (Poly Chapitre3 T27) pour simplifier le code :

$$\text{La mise à jour de C.G. Broyden, D. Goldfarb et D.F. Shanno (BFGS) est définie par: } H_k = H_{k-1} + \frac{\overline{y}_{k-1} \overline{y}_{k-1}^T}{\overline{d}_{k-1}^T \overline{d}_{k-1}} - \frac{H_{k-1} \overline{d}_{k-1} \overline{d}_{k-1}^T H_{k-1}}{\overline{d}_{k-1}^T H_{k-1} \overline{d}_{k-1}}$$

Voici les résultats obtenus pour les différents beta1 et beta2 :

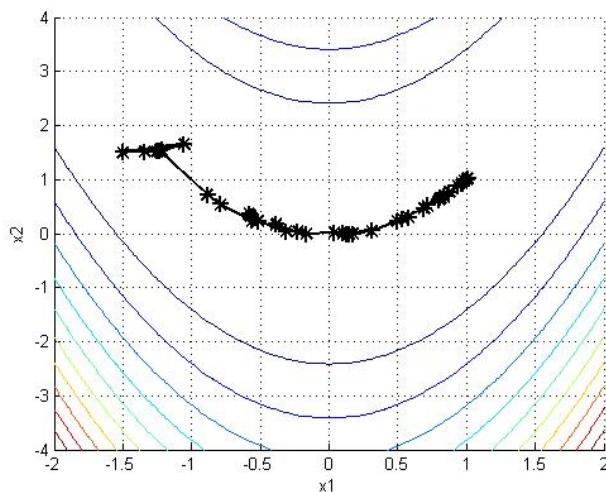
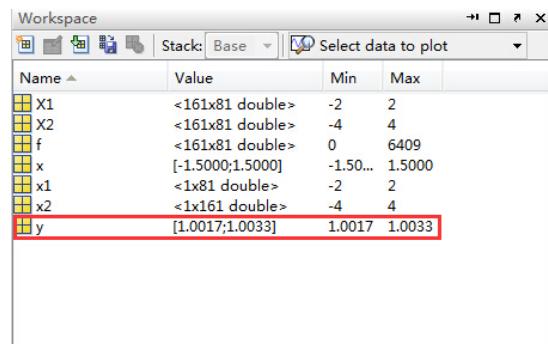
### CONDITION 1 :

$x=[-1.5; 1.5]; \beta_1=0.1 \beta_2=0.9 n=40$



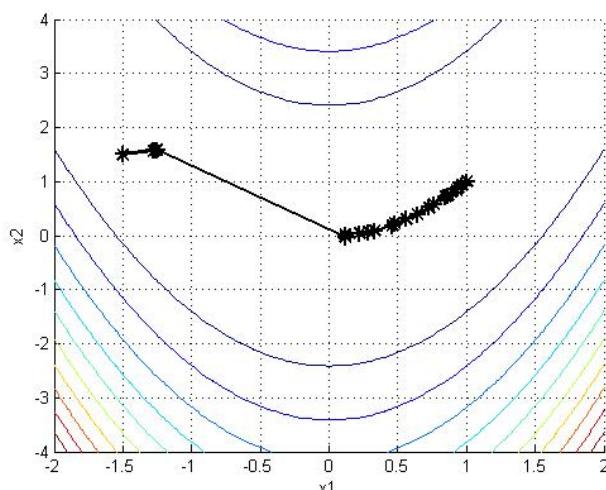
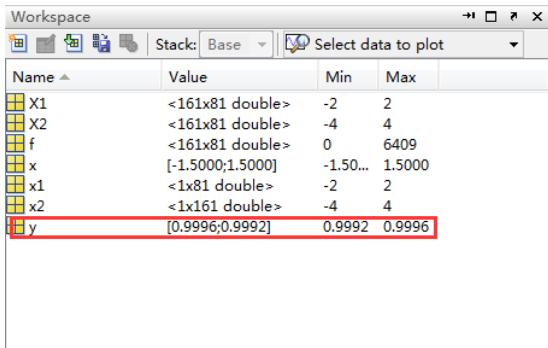
### CONDITION 2 :

$X=[-1.5; 1.5]; \beta_1=0.1 \beta_2=0.5 n=36$



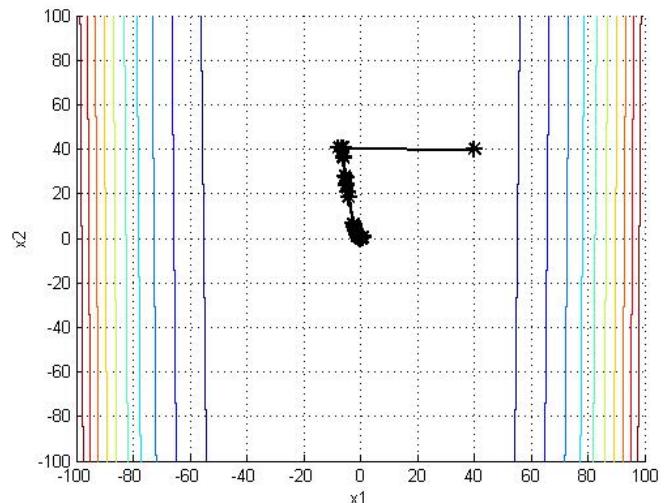
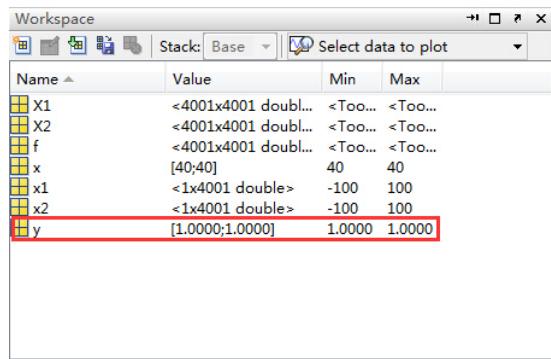
### CONDITION 3 :

$X=[-1.5; 1.5]; \beta_1=0.4 \beta_2=0.7 n=23$



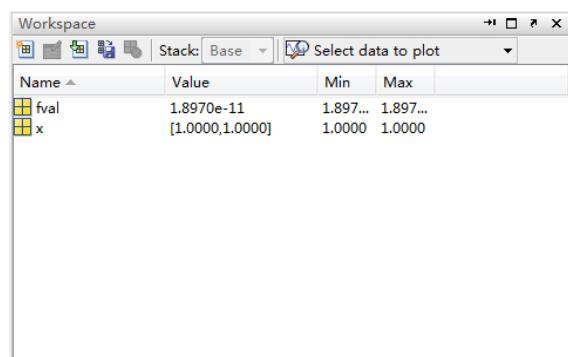
#### CONDITION 4 :

X=[40;40]; beta1=0.1 beta2=0.5 n=62



Pour la condition 4, même si le point de départ est très loin du point de convergence, il peut encore converger.

#### 2. En utilisant la fonction fminunc de la Toolbox Matlab Optimisation



```

Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the function tolerance.

stopping criteria details

x =
1.0000    1.0000

fval =
1.8970e-011

```

x =

1.0000 1.0000

fval =

1.8970e-011

Avec fminunc il trouve le minimum plus rapidement que toutes les méthodes réalisées précédemment et avec une plus grande précision.

### 3. Comparer les différences avec la méthode de Newton

Différences :

La méthode quasi-Newton est plus efficace que la méthode de Newton et la méthode des plus fortes pentes. En général, il faut moins d'itérations pour arriver au plus minimum (1,1).

Pour :

$\beta_1=0.1, \beta_2=0.9$ , il faut 40 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01. (Newton : 91 itérations).

$\beta_1=0.1, \beta_2=0.5$ , il faut 36 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

(Newton : 20 itérations).

$\beta_1=0.4, \beta_2=0.7$ , il faut 23 itérations pour arriver au point min (1,1) avec une marge d'erreur de 0.01.

(Newton : 18 itérations).

Ces différences s'expliquent par le fait qu'il faut calculer la dérivée de la fonction pour chaque itération pour la méthode de Newton avec recherche linéaire, le temps de calcul et la complexité sont donc plus importants.

Pour la méthode quasi-Newton BFGS, on remplace la matrice Hessienne par son approximation BFGS  $\nabla^2 f(x) \rightarrow H_k$  (forcément définie positive). Cela permet de réduire les temps de calcul.

Les variations des  $\beta_1$  et  $\beta_2$  influencent peu sur la méthode quasi-Newton BFGS.

## Question5

La cantine à Télécom Saint-Etienne fournit deux menus chaque déjeuner : un est un menu steak-frites et l'autre est un plat du jour. Il lui est indiqué de vendre ses deux menus à €3 et €3.5.

Les coûts de production de la cantine sont respectivement de €1 et €1.2. Le coût fixe de chaque déjeuner est de €50. Le nombre de vente de menu aura une influence sur le prix moyen du menu à cause des marchés concurrentiels. On estime que pour chaque menu vendu, le prix moyen diminue de un pour cent. La vente des menus steak-frites diminue de cinq pour mille le prix du menu du jour

et la vente des menus du jour diminue de dix pour mille le prix du menu steak-frites.

La cantine cherche donc à maximiser ses bénéfices en produisant le bon nombre de menus du jour et de menus steak-frites.

**Résolution :**

Variables :

$x$  = quantité de menus steak-frites vendus

$y$  = quantité de menus du jour vendus

$p$  = prix de vente moyen d'un menu steak-frites

$q$  = prix de vente moyen d'un menu du jour

$C$  = coûts de production total

$R$  = revenus des ventes de menus

$P$  = profits des ventes de menus

Constantes :

Prix initial des deux menus: €3 et €3.5

Coûts de fabrication à l'unité : €1 et de €1.2

A chaque vente de l'un des menus, le prix de vente moyen diminue de  $a= €0.01$  (élasticité-prix),

Coefficients d'interaction : €0.005 et €0.01.

Coûts fixes : €50

Relation :

1. Pour chaque type de menu vendu, le prix de vente moyen de ce menu va baisser de 0.01.
2. On estime que pour chaque menu du jour vendu, le prix de vente d'un menu steak-frites va diminuer de 0.005, et pour chaque menu steak-frites vendu, le prix de vente moyen d'un menu du jour va diminuer de 0.01.

Donc :

Prix de vente d'un menu steak-frites :  $p=3-0.01*x-0.005*y$  ;

Prix de vente d'un menu du jour :  $p=3.5-0.01*y-0.01*x$  ;

Revenus :  $R=p*x+q*y$  ;

Les coûts de production :  $C=50+1*x+1.2*y$  ;

Profits :  $P=R-C$  ;

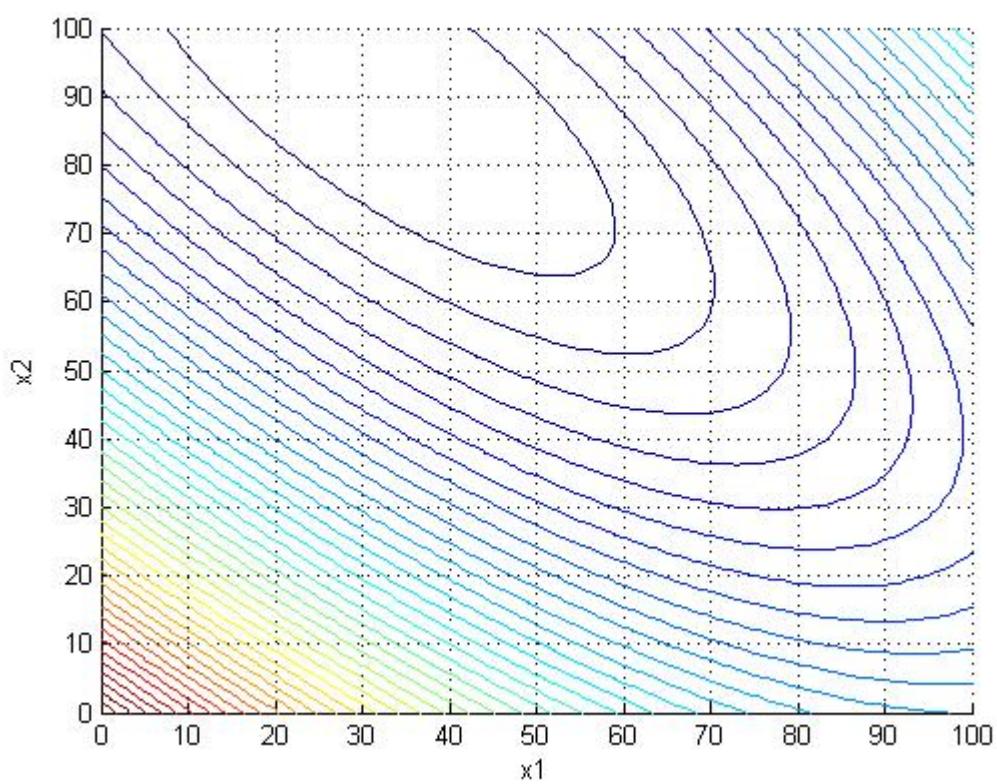
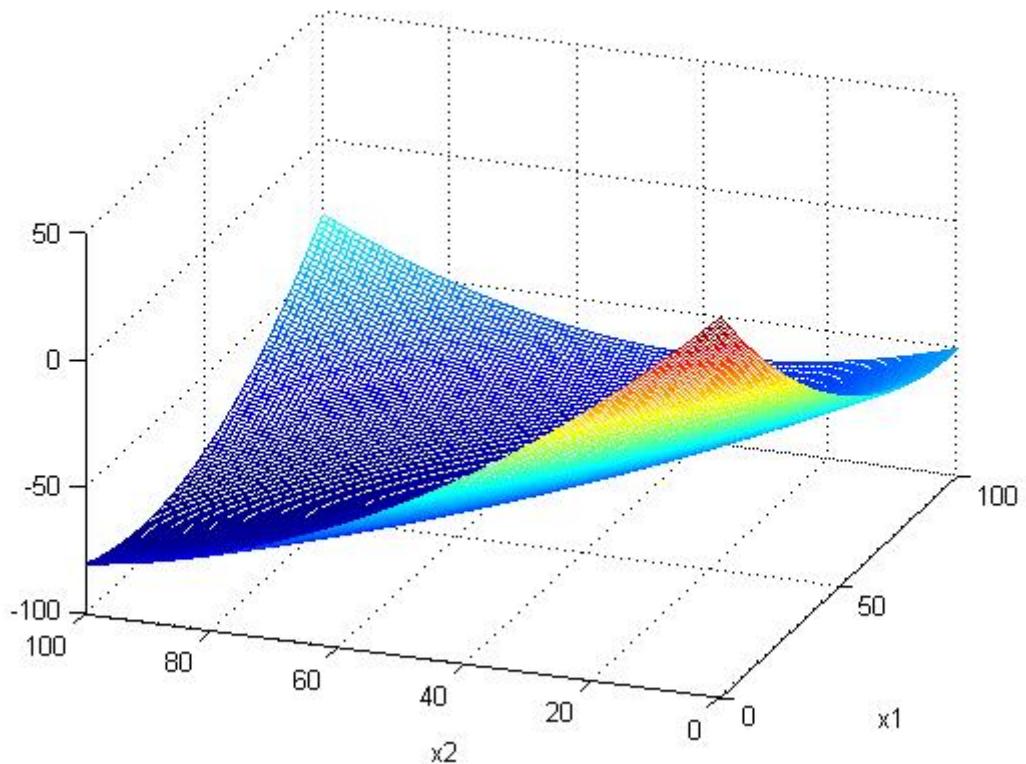
Donc le profit doit être :

$$\text{Max } P(x,y) = (3-0.01*x-0.005*y)*x + (3.5-0.01*y-0.01*x)*y - (50+1*x+1.2*y)$$

$$= 2x + 2.3*y - 0.01*x^2 - 0.01*y^2 - 0.015*x*y - 50$$

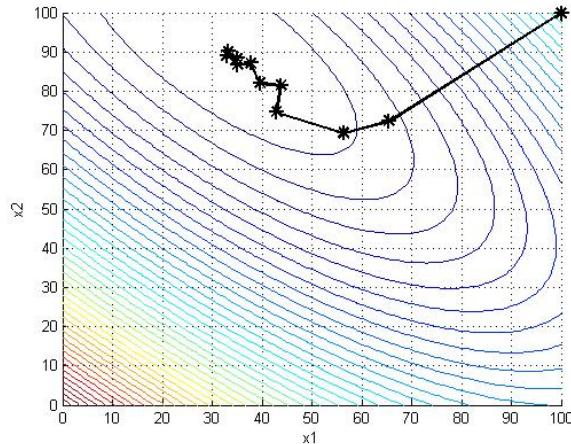
Pour utiliser les méthodes analysées avant, on cherche le minimum:

$$\text{Min } P(x,y) = -2x - 2.3y + 0.01*x^2 + 0.01*y^2 + 0.015*x*y + 50$$



Avec la méthode des plus fortes pentes avec préconditionnement :

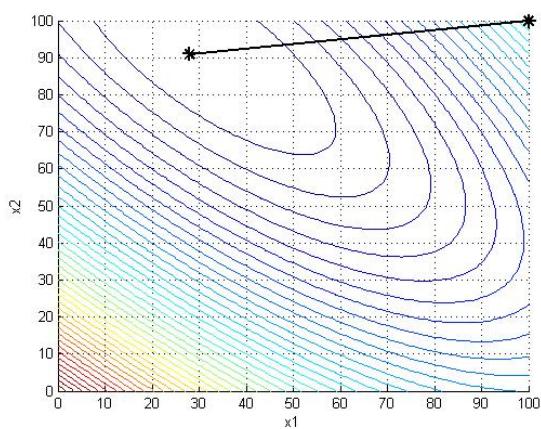
[100 ;100] beta1=0.4 beta2=0.7 n=11



Workspace			
Name	Value	Min	Max
X1	<101x101 double>	0	100
X2	<101x101 double>	0	100
f	<101x101 double>	-86.5...	50
fval	-86.5714	-86.5...	-86.5...
x	[100;100]	100	100
x1	<1x101 double>	0	100
x2	<1x101 double>	0	100
y	[32.5701;90.1026]	32.57...	90.10...

Avec la méthode de Newton avec recherche linéaire

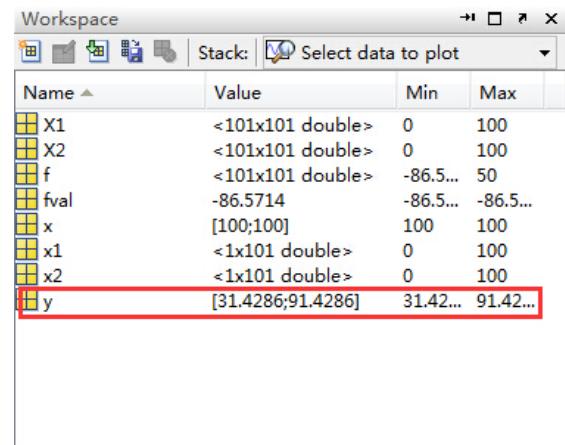
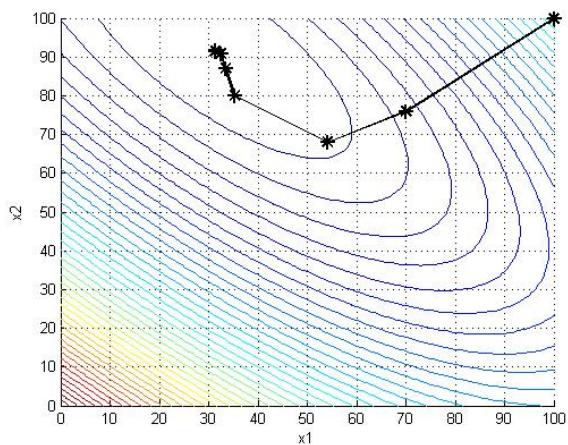
[100 ;100] beta1=0.4 beta2=0.7 n=2



Workspace			
Name	Value	Min	Max
X1	<101x101 double>	0	100
X2	<101x101 double>	0	100
f	<101x101 double>	-86.5...	50
fval	-86.5714	-86.5...	-86.5...
x	[100;100]	100	100
x1	<1x101 double>	0	100
x2	<1x101 double>	0	100
y	[31.6000;91.4500]	31.60...	91.45...

Avec La méthode de quasi-Newton BFGS

[100 ;100] beta1=0.4 beta2=0.7 n=13



Avec la fonction fminunc de la Toolbox Matlab Optimisation

Dans le fichier fmi.m :

```
function f = fmi(x)
f = -2*X1-2.3*X2+0.01*X1.^2+0.01*X2.^2+0.015*X1.*X2+50;
```

```
>> Test_fminunc
Warning: Gradient must be provided for trust-region algorithm;
    using line-search algorithm instead.
> In fminunc at 365
  In Test_fminunc at 3

Local minimum found.

Optimization completed because the size of the gradient is less than
the default value of the function tolerance.

<stopping criteria details>

x =
31.4285  91.4286

fval =
-86.5714
```

Conclusion :

Les quatre méthodes ont trouvé le même minimum. (31.4285, 91.4286)

Donc, la cantine doit vendre 32 menus steak-frites et 92 menus du jour chaque déjeuner pour obtenir le maximum de profit. Le bénéfice net est de €86.57 pour chaque déjeuner.